02/17/2022

Chapter 1: Getting Started

In this course, we will use onlineGDB as the main Integrated Development Environment (IDE). Here is the url to it: https://www.onlinegdb.com/#

1. Hello World!

Every C++ program must have exactly one global function named `main()`. The program starts by executing that function. The `int` value returned by `main()`, if any, is the program's return value to "the system." If no value or 0 is returned, the system will receive a value indicating successful completion. A non-zero value from `main()` indicates failure.

Typically, a program produces some output. Here is a program that writes `Hello World!`

Hello.cpp

```
#include <iostream>

int main(){
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

The line `#include<iostream>` instructs the compiler to include the declarations of the *standard stream I/O facilities* defined in `iostream`. Without these declarations, the expression

```
std::cout << "Hello World!" << std::endl;
```

will make no sense to the computer. The operator `<<` ("put to") writes its second argument/operand onto its first. In this case, the string literal `"Hello World!"` is written onto the standard output stream `std::cout`. A string literal is a sequence of characters surrounded by double quotes. The `std::` specifies that the name `cout` belongs to the standard-library (std) *namespace*. The `endl` indicates an end-of-line.

If you need to get input from the user through the standard input stream, you can use the following syntax which is also part of the *standard stream I/O facilities*. The `cin` is the standard input, the operator >> is an extraction operator.

```
int i;
std::cout << "Please input an integer: ";
```

```
std::cin >> i;
std::cout << "The value of your input: " << i << std::endl;
```

2. A Hello World Program with a Class and an Object

In the following example, we define a class named `Hello` in a .h file (a header file, introduced in more details in the next section), implement the details of the class in a .cpp file, then use the class in the `main()` in another .cpp file.

hello.h

```
#ifndef HELLO_H
#define HELLO_H
namespace hello{
 class Hello{
     public:
     void helloWorld();
 };
}
#endif
```

(Class Member Function) A member function is a function that is defined by a class. Member functions are defined once for the class but are treated as members of each object. In the previous lines of code, the `helloWorld()` is a member function declared in the .h file of class `Hello`. The member function's definition is placed in the corresponding .cpp file:

hello.cpp

```
#include <iostream>
#include "hello.h"

namespace hello{
   void Hello::helloWorld(){
      std::cout<<"Hello World!! C++!!!"<<std::endl;
   }
}
```

A dot operator (.) is used to call a member function:

first.cpp

```
#include "hello.h"

int main(){
   hello::Hello h;
   h.helloWorld();
```

```
    return 0;
}
```

Output:

Hello World!! C++!!!

Header Files (.h)

In order to ensure that the class definition is the same in each file, classes are usually defined in header files. Typically, classes are stored in headers whose name derives from the name of the class. For example, the `string` library type is defined in the `string` header. Similarly, as we've already seen, we defined our `Hello` class in a header file named `hello.h`.

The most common technique for making it safe to include a header multiple times relies on the preprocessor. The preprocessor—which C++ inherits from C—is a program that runs before the compiler to modify our source. Our program already rely on one preprocessor facility, `#include`. When the preprocessor sees `#include`, it replaces the `#include` with the contents (lines of code) of the specified header (such as `iostream`).

C++ programs also use the preprocessor to define header guards. Header guards rely on preprocessor variables. **Preprocessor variables have one of two possible states: defined** or **not defined.** The `#define` directive takes a name and defines that name as a preprocessor variable.

There are two other directives that **test** whether a given preprocessor variable has or has not been defined:

1. `#ifdef` is true if the variable has been defined, and
2. `#ifndef` is true if the variable has not been defined.

If the test is true, then everything following the directive (`#ifdef` or `#ifndef)` is processed up to the matching `#endif`. See the definition of the hello.h file as an example.

3. Some more examples using C++

3.1. Conditional statements[1]

Syntax:

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

Example 1 checking a single condition:

```
#include <iostream>

using namespace std;

int main() {
    int i = 10;

    if (i > 15)
    {
       cout << "10 is less than 15" << endl;
    }

    cout << "I am Not in if";
}
```

Example 2 two-way conditional check:

```
#include <iostream>

using namespace std;

int main() {
    int i = 20;

    if (i < 15){

        cout << "i is smaller than 15";
    }
    else{

        cout << "i is greater than 15";
    }
    return 0;
}
```

Example 3 multiple conditions being tested:

```
#include <iostream>

using namespace std;

int main() {
    int i = 20;
```

[1] https://www.geeksforgeeks.org/decision-making-c-c-else-nested-else/

```
    if (i == 10)
        cout << "i is 10";
    else if (i == 15)
        cout << "i is 15";
    else if (i == 20)
        cout << "i is 20";
    else
        cout << "i does not match any condition";
}
```

3.2. Logical operators: and (`&&`), or (`||`), not (`!`)[2].

```
#include <iostream>

using namespace std;

int main() {
    if(4 != 5 && 4 < 5)     // true
        cout << "true!" << endl;
}
```

3.3. Using the `for` loop in C++[3].

Syntax:

```
for (initialization expr; test expr; update expr)
{
     // body of the loop
     // statements we want to execute
}
```

Example: sum the numbers from 0 to 9 using the `for` loop and print the sum.

<u>Sum.cpp</u>

```
#include <iostream>

using namespace std;

int main()
{
    int sum = 0;
    for (int i = 0; i < 10; ++i)
        sum += i;
    cout << "Sum is: " << sum << endl;
    return 0;
}
```

Note: `++i` means `i = i + 1`; the increment taking effect on the same line.

3.4. Using the `while` loop in C++.

[2] https://www.geeksforgeeks.org/operators-c-c/

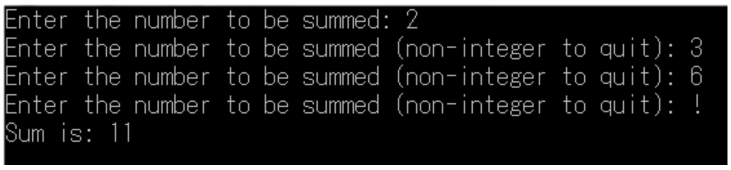
[3] https://www.geeksforgeeks.org/loops-in-c-and-cpp/

Syntax:

```
initialization expression;
while (test_expression)
{
   // statements

  update_expression;
}
```

Example: Let's ask the user to input a set of numbers to sum. In this case, we won't know how many numbers to add. Instead, we'll keep reading numbers until there are no more numbers to read. Write a program and use `while` loop for the task.



```
Enter the number to be summed: 2
Enter the number to be summed (non-integer to quit): 3
Enter the number to be summed (non-integer to quit): 6
Enter the number to be summed (non-integer to quit): !
Sum is: 11
```

AddSum.cpp

A:

```
#include <iostream>

using namespace std;

int main()
{
    int sum = 0;
    int num;
    cout << "Enter the number to be summed: ";
    while (cin >> num) {
        sum += num;
        cout << "Enter the number to be summed (non-integer to
quit): ";
    }
    cout << "Sum is: " << sum << endl;
    return 0;
}
```

Q: What is the effect to use an `istream` as a condition?

A:

– When we use an `istream` as a condition, the effect is to test the state of the stream. If the stream is valid (still possible to read another input), then the test succeeds.

– An `istream` becomes invalid when we:

a. hit end-of- file (for file input) or

b. encounter an invalid input, such as reading a value that is not an integer.

An `istream` that is in an invalid state will cause the condition to become false.

3.5. The `break` and the `continue` in loops

`break`: terminates the smallest enclosing loop (i.e., `while`, `do-while`, `for` or `switch` statement).

`continue`: skips the rest of the loop statement and causes the next iteration of the loop to take place.

Example:

```
#include <iostream>
using namespace std;
main()
{
    int i;
    cout << "The loop with break produces output as: ";

    for (i = 1; i <= 5; i++) {

        // Program comes out of loop when
        // i becomes multiple of 3.
        if ((i % 3) == 0)
            break;
        else
            cout << i << " ";
    }

    cout << endl << "The loop with continue produces output as: 
";
    for (i = 1; i <= 5; i++) {

        // The loop prints all values except
        // those that are multiple of 3.
        if ((i % 3) == 0)
            continue;
         cout << i << " ";
    }
}
```

Output:

```
The loop with break produces output as: 1 2
The loop with continue produces output as: 1 2 4 5
```

Please work on lab problems here: https://oop.tanjimeow.com/

HW will be issued soon, and due in one week.